PCS 2428 / PCS 2059
Inteligência Artificial

Prof. Dr. Jaime Simão Sichman
Prof. Dra. Anna Helena Reali Costa

Sistemas Lógicos

Sumário

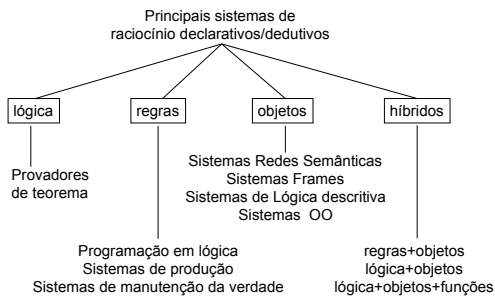
1. Introdução
2. Sistemas de Produção
3. Sistemas Baseados em Lógica
4. Redes Semânticas
5. Frames
6. Sistemas de Manutenção da Verdade (TMS)
7. Conclusões

Sumário

1. Introdução
2. Sistemas de Produção
3. Sistemas Baseados em Lógica
4. Redes Semânticas
5. Frames
6. Sistemas de Manutenção da Verdade (TMS)
7. Conclusões

Introdução

- Agentes podem ser formalizados e implementados por sistemas baseados em conhecimento
  - Modularidade
  - Controle isolado do conhecimento
  - Facilidade de prototipagem
- Sistemas com finalidades distintas
  - Sistemas de Produção
  - Provadores de Teoremas e Linguagens de Programação Lógica
  - Sistemas de Frames e Redes Semânticas
  - Sistemas Baseados em Lógica Descritiva

Sistemas Baseados em Conhecimento

Principais sistemas de
raciocínio declarativos/dedutivos

- lógica
  - Provadores de teorema
- regras
  - Programação em lógica
  - Sistemas de produção
  - Sistemas de manutenção da verdade
- objetos
  - Sistemas Redes Semânticas
  - Sistemas Frames
  - Sistemas de Lógica descritiva
  - Sistemas OO
- híbridos
  - regras+objetos
  - lógica+objetos
  - lógica+objetos+funções

Sumário

1. Introdução
2. Sistemas de Produção
3. Sistemas Baseados em Lógica
4. Redes Semânticas
5. Frames
6. Sistemas de Manutenção da Verdade (TMS)
7. Conclusões

Regras de Produção

- Características:
  - Representam conhecimento de forma modular
    - cada regra representa um "pedaço" de conhecimento independente
    - cuidado: a consistência deve ser mantida.
  - São fáceis de compreender (legíveis) e de modificar
  - Novas regras podem ser facilmente inseridas na BC
  - Podem ser usadas tanto com raciocínio progressivo quanto com raciocínio regressivo.

Sistemas de Produção

- São sistemas baseados em regras de produção
- Consistem em 3 módulos principais:
  - A Base de Regras: permanente
    - regras se-então e fatos conhecidos
  - A Memória de Trabalho: temporária
    - base de fatos derivados durante a "vida" do agente
    - percepções do agente e fatos gerados a partir da BR pelo mecanismo de inferência
  - O Mecanismo de Inferência (máquina de inferência)
    - determina o método de raciocínio utilizado (progressivo ou regressivo)
    - utiliza estratégias de busca com casamento (unificação)
    - resolve conflitos e executa ações.

Arquitetura dos Sistemas de Produção

Base de Regras

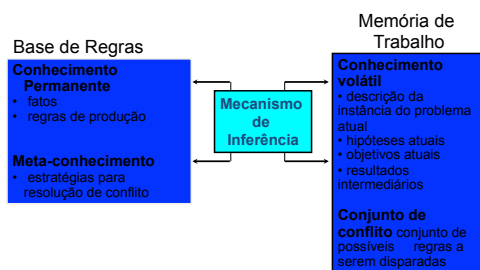
Conhecimento Permanente
- fatos
- regras de produção

Meta-conhecimento
- estratégias para resolução de conflito

Mecanismo de Inferência

Memória de Trabalho

Conhecimento volátil
- descrição da instância do problema atual
- hipóteses atuais
- objetivos atuais
- resultados intermediários

Conjunto de conflito conjunto de possíveis regras a serem disparadas

Exemplo de Base de Regras

- Bicicleta: Se **veículoTipo=ciclo**
  E **num-rodas=2**
  E **motor=não**
  Então **veículo=*Bicicleta***
- Triciclo: Se **veículoTipo=ciclo**
  E **num-rodas=3**
  E **motor=não**
  Então **veículo=*Triciclo***
- Motocicleta: Se **veículoTipo=ciclo**
  E **num-rodas=2**
  E **motor=sim**
  Então **veículo=*Motocicleta***

Exemplo de Base de Regras

- CarroSport: Se **veículoTipo=automóvel**
  E **tamanho=pequeno**
  E **num-portas=2**
  Então **veículo=*CarroSport***
- Sedan: Se **veículoTipo=automóvel**
  E **tamanho=médio**
  E **num-portas=4**
  Então **veículo=*Sedan***
- MiniVan: Se **veículoTipo=automóvel**
  E **tamanho=médio**
  E **num-portas=3**
  Então **veículo=*MiniVan***

Exemplo de Base de Regras

- UtilitárioSport: Se **veículoTipo=automóvel**
  E **tamanho=grande**
  E **num-portas=4**
  Então **veículo=*UtilitárioSport***
- Ciclo: Se **num-rodas<4**
  Então **veículoTipo=*ciclo***
- Automóvel: Se **num-rodas=4**
  E **motor=sim**
  Então **veículoTipo=*automóvel***

Encadeamento Progressivo

- Dos dados à conclusão - *data-driven inference*
  - Parte dos fatos na Base de Regras e na Memória de Trabalho, buscando quais regras eles satisfazem, para produzir assim novas *conclusões* (fatos) e/ou realizar *ações.*
- Três etapas:
  - Busca, Casamento (unificação), Resolução de conflito
- É uma estratégia de inferência muito rápida
  - usada em sistemas de monitoramento e diagnóstico em tempo real.
- Ferramentas comerciais que implementam esta estratégia
  - OPS5, OPS85, IBM: TIRS

Encadeamento Progressivo: Algoritmo

1. **Armazena as regras da BR na máquina de inferência (MI) e os fatos na memória de trabalho (MT);**
2. **Adiciona os dados iniciais à memória de trabalho;**
3. **Compara o antecedente das regras com os fatos na MT. As regras cujo antecedente "casa" (unifica) com esses fatos podem ser disparadas (*conjunto de conflito);***
4. **Usa o procedimento de resolução de conflito para selecionar uma única regra desse conjunto;**
5. **Dispara a regra selecionada e verifica o seu conseqüente:**
   a) se for um fato, atualiza a MT
   b) se for uma ação, chama o procedimento que ativa os efetuadores do agente e atualiza a MT
6. **Repete os passos 3, 4 e 5 até que o conjunto de conflito se torne vazio.**

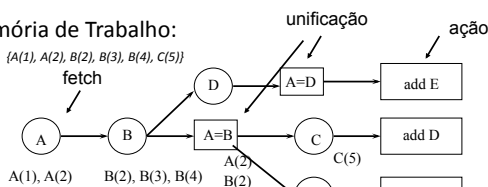
Encadeamento Progressivo: Busca e Casamento

- Busca
  - Se a BR é muito grande, verificar todas as regras gasta muito tempo
  - Prioridade:
    - regras cujo antecedente se refere a um fato recentemente inserido na MT (pela última regra disparada, por exemplo)
- Casamento (unificação)
  - O antecedente de cada regra selecionada é comparado com os fatos na MT usando **busca em largura**
  - ex.: fatos e regra sobre automóveis
    - ***MT1:*** *veloz(Kadet-2.0), veloz(BMW), veloz(Gol-2.0), veloz(Mercedes), importado(BMW), importado(Mercedes)*
    - ***BC: Se*** *veloz(x)* ***e*** *importado(x)* ***então*** *caro(x)*
    - ***MT2:*** *MT1 + {caro(BMW), caro(Mercedes)}*

Encadeamento Progressivo: Busca e Casamento

- Casamento (unificação)
  - A forma mais simples de realizar unificação é ineficiente
  - Exemplo: 100 fatos na memória de trabalho, 200 regras com 5 antecedentes cada, 1000 ciclos para resolver o problema, deveria realizar 10^7 unificações!
- Como solução, temos o Algoritmo RETE (rede)
  - compila a memória de regras
  - cria uma rede de dependências entre as regras da BR
  - minimiza o número de testes requeridos durante a fase de casamento
  - elimina duplicações entre regras

Encadeamento Progressivo: Algoritmo RETE

- Base de Regras:
  - *A(x) ^ B(x) ^ C(y) => add D(x)*
  - *A(x) ^ B(y) ^ D(x) => add E(x)*
  - *A(x) ^ B(x) ^ E(z) => delete A(x)*
- Memória de Trabalho:
  - *{A(1), A(2), B(2), B(3), B(4), C(5)}*

Encadeamento Progressivo: Algoritmo RETE

- Elimina duplicações entre regras, pois no exemplo as 3 regras utilizam uma conjunção dos predicados A e B
- Na maior parte dos casos, os sistemas de produção alteram apenas poucos fatos na BC, assim a maior parte dos testes feitos no ciclo i terão o mesmo resultado no ciclo i+1
- Deve ser atualizada sempre que um fato for adicionado ou retirado da BC, mas o custo desta alteração é pequeno

Encadeamento Progressivo: Resolução de Conflitos

- Resolução de conflitos
  - heurística geral para escolher um subconjunto de regras a disparar
- Exemplos:
  - **Não duplicação:** não executar a mesma regra com os mesmos argumentos duas vezes.
  - **Prioridade de operação:** preferir ações com prioridade maior (~ sistemas ação-valor - LPO).
  - **Recency (“*recenticidade*”):** preferir regras que se referem a elementos da Memória de Trabalho criados recentemente.
  - **Especificidade**: preferir regras que são mais específicas.

Encadeamento Progressivo: Exemplo

- Carregar a BR de veículos na MI e atribuir valores iniciais para algumas variáveis, guardando esses fatos na MT.

  Fatos iniciais:
  - num-rodas=4
  - motor=sim
  - num-portas=3
  - tamanho=médio
- Fase de “casamento”
  - Conjunto de conflito da 1a rodada de inferência resulta em apenas uma regra

    Automóvel: Se num-rodas=4
E motor=sim
Então *veículoTipo=automóvel*

Encadeamento Progressivo: Exemplo

- A resolução de conflito fica então trivial.
- Fatos na MT:
  - num-rodas=4; motor=sim; num-portas=3; tamanho=médio
  - *veículoTipo=automóvel*
- Casamento: **segunda rodada de inferência seleciona apenas 1 regra para o conjunto de conflito:**
  - **MiniVan: Se** veículoTipo=automóvel
E tamanho=médio
E num-portas=3
Então *veículo=MiniVan*

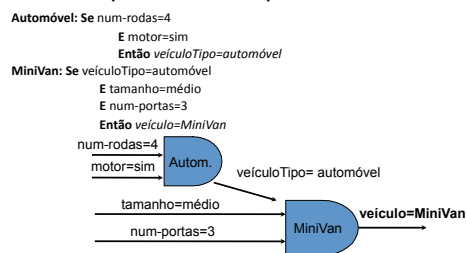
Encadeamento Progressivo: Exemplo

- Fatos na MT:
  - num-rodas=4; motor=sim; num-portas=3; tamanho=médio
  - veículoTipo=automóvel; *veículo=MiniVan*
- Casamento:
  - terceira rodada de inferência seleciona a mesma regra que na rodada anterior
  - como esta já foi disparada, não será adicionada novamente ao conjunto de conflito
  - com o conjunto de conflito vazio, o processo de inferência pára
- Com os fatos na MT, concluímos então que o veículo procurado é uma *Minivan*.

Encadeamento Progressivo: Disparo das Regras

- **O fluxo de informações se dá através de uma série de regras encadeadas a partir das assertivas para as conclusões**

  Automóvel: Se num-rodas=4
E motor=sim
Então *veículoTipo=automóvel*

  MiniVan: Se veículoTipo=automóvel
E tamanho=médio
E num-portas=3
Então *veículo=MiniVan*

Encadeamento Regressivo

- Da hipótese aos dados - *goal-directed inference*
  - Parte da *hipótese* que se quer provar, procurando regras na BR cujo *conseqüente* satisfaz essa hipótese.
  - usa as regras da BR para responder a perguntas
  - busca provar se uma asserção é verdadeira
    - ex.: criminoso(West)?
  - só processa as regras relevantes para a pergunta
- Duas etapas:
  - Busca e Casamento (unificação)
- Utilizado em sistemas de aconselhamento
  - trava um “diálogo” com o usuário
  - ex.: MYCIN

Encadeamento Regressivo: Algoritmo

1. Armazena as regras da BC na máquina de inferência (MI) e os fatos na memória de trabalho (MT);

2. Adiciona os dados iniciais à memória de trabalho;

3. Especifica uma variável *"objetivo"* para a MI;

4. Busca o conjunto de regras que se referem à variável objetivo no conseqüente da regra.

- Isto é, seleciona todas as regras que atribuem um valor à variável objetivo quando disparadas.

Insere cada regra na pilha de objetivos;

Encadeamento Regressivo: Algoritmo

5. Se a pilha de objetivos está vazia, pare.

6. Selecione a regra no topo da pilha;

7. Tente provar que essa regra é verdadeira testando, um a um, se todos os seus antecedentes são verdadeiros:

a) se o 1o. antecedente é V, vá em frente para o próximo

b) se ele for F, desempilhe essa regra e volte ao passo 5

c) se o seu valor-verdade é desconhecido porque a variável do antecedente é desconhecida, vá para o passo 4 com essa variável como variável objetivo

d) se todos os antecedentes são V, dispare a regra, instancie a variável no conseqüente para o valor que aparece nessa regra, retire a regra da pilha e volte ao passo 5.

Encadeamento Regressivo: Busca e Casamento

- Busca e Casamento
  - O sistema percorre a BC em busca regras cujo conseqüente "casa" com a hipótese de entrada
  - Se a hipótese de entrada é um fato, a busca pára quando encontra a 1a regra que casa com ele, e o sistema devolve uma variável booleana (**V** ou **F**).
  - Se a hipótese tem alguma variável livre (não instanciada), o sistema (programador) pode optar por devolver a 1a instanciação encontrada, ou por devolver uma lista com todas as possíveis instâncias para aquela variável.
  - Portanto, não há conflito de execução de regras
  - Unificação é realizada com busca em profundidade
    - exs.: veículo=MiniVan?, criminoso(West)?

Encadeamento Regressivo: Exemplo

- Carregar a BR de veículos na MI e os fatos na MT
- Fatos iniciais:
  - num-rodas=4, motor=sim, num-portas=3, tamanho=médio
- Especificar variável objetivo
  - veículo=?
- Pilha de objetivos
  - regras com variável objetivo no conseqüente
    - as 7 primeiras regras da nossa BC

Encadeamento Regressivo: Exemplo

- Tenta provar verdadeiros os antecedentes da 1a regra usando busca em profundidade
  - **Bicicleta: Se** veículoTipo=ciclo
    E num-rodas=2
    E motor=não
    Então veículo=*Bicicleta*
- *VeículoTipo=ciclo* não aparece na MT
  - nova variável objetivo
- Atualiza pilha de objetivos
  - inclui regras com nova variável objetivo no conseqüente
    - apenas a penúltima regra da nossa BC

Encadeamento Regressivo: Exemplo

- *veículoTipo=ciclo* só é verdade em apenas uma regra
  - **Ciclo: Se** num-rodas < 4
    Então veículoTipo=*ciclo*
- Verifica o valor verdade dos antecedentes da regra
  - num-rodas < 4 ===> FALSO!
- Donde se deduz que *veículo=Bicicleta* é Falso!

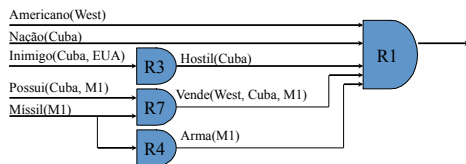
Encadeamento Regressivo: Exemplo

- Se o fato a ser provado não aparece explicitamente na base e nem pode ser deduzido por nenhuma outra regra...
- ... duas coisas podem ocorrer, dependendo da implementação do sistema
  - o fato é considerado FALSO
    - ex. Prolog
  - o sistema consulta o usuário via sua interface
    - ex. ExpertSinta

Encadeamento Regressivo: Exemplo

- Desempilha as outras regras, uma a uma, até encontrar a regra abaixo - que vai dar certo!
  - **MiniVan: Se** veículoTipo=automóvel
    E tamanho=médio
    E num-portas=3
    Então veículo=*MiniVan*
- *VeículoTipo=automóvel* não existe na MT
  - **Automóvel: Se** num-rodas=4 OK! (1)
    E motor=sim OK! (2)
    Então veículoTipo=*automóvel* *===> OK! (3)*
- Tenta provar os outros antecedentes da regra, que estão todos instanciados na MT, e são verdadeiros!
- veículo=*MiniVan* é verdade!

Encadeamento Progressivo: Disparo das Regras

Americano(West)
Nação(Cuba)
Inimigo(Cuba, EUA)
Possui(Cuba, M1)
Missil(M1)
R3
Hostil(Cuba)
R7
Vende(West, Cuba, M1)
R4
Arma(M1)
R1

Tipos de Regras

- Regras causais (dedução)
  - assumem causalidade
    - algumas propriedades escondidas no mundo causam a geração de certas percepções
  - Se "causa" ocorrer
    então "conseqüência" ocorre
    - **se** há fogo, **então** há fumaça
    - **se** chove, **então** a grama fica molhada
- Sistemas que raciocinam com regras causais são conhecidos como Sistemas Baseados em Modelos

Tipos de regras

- Regras de diagnóstico (abdução)
  - Supõe a presença de propriedades escondidas a partir das percepções do agente.
  - Se "conseqüência" ocorreu
    então "causa" deve ser...
    - **se** há fumaça,**então** conclui-se que há fogo
    - **se** a grama está molhada, **então** conclui-se que o aguador ficou ligado
- Problema:
  - Raciocínio abdutivo preserva a falsidade, mas não a verdade
- Sistemas que raciocinam com regras de diagnóstico são conhecidos como Sistemas Baseados em Diagnóstico

Tipos de regras

- A distinção entre raciocínio baseado em modelos e raciocínio baseado em diagnóstico é importante.
- É perigoso misturar esses dois tipos de regras numa mesma BC!!!
  - se choveu é porque o aguador estava ligado
- O raciocínio baseado em modelos tem crescido na preferência:
  - ex. diagnóstico médico e de falhas em equipamentos

Regras com Fator de Incerteza

- Na maioria dos sistemas *reais*, é necessário associar-se um *fator de incerteza* (ou de *confiança*) a algumas regras na BR
  - As regras que se aplicam em 100% dos casos são poucas...
- Incerteza nos dados e na aplicação das regras
  - *If* (previsão-do-tempo = chuva) > 80%
    and (previsão-períodos-anteriores = chuva) = 85%
    then (chance-de-chuva = alta) = 90%
- Deve-se combinar as incertezas dos antecedentes
  - teoria da probabilidade?
  - "senso-comum"?
  - experiência do especialista na área?

Sistemas de Produção : Vantagens e Limitações

- Vantagens
  - As regras são de fácil compreensão.
  - Inferência e explicações são facilmente derivadas.
  - Manutenção é relativamente simples, devido a modularidade.
  - "Incerteza" é facilmente combinada com as regras.
  - Cada regra é (normalmente) independente das outras.
  - **São mais eficientes que os sistemas de programação em lógica, embora menos expressivos**
- Desvantagens
  - Conhecimento complexo requer muitas (milhares de) regras.
  - Esse excesso de regras cria problemas para utilização e manutenção do sistema.
  - Não é robusto.

Sumário

1. Introdução
2. Sistemas de Produção
3. Sistemas Baseados em Lógica
4. Redes Semânticas
5. Frames
6. Sistemas de Manutenção da Verdade (TMS)
7. Conclusões

Sistemas Baseados em Raciocínio Lógico

- Implementação de sentenças e fatos
  - c: P(x) ∧ Q(x) pode ser implementado como um tipo de dados onde : op[c] = ∧ e args[c] = [P(x), Q(x)]
- Para realizar as operações de store e fetch, precisamos de um mecanismo eficiente para recuperar e armazenar as cláusulas
  - 1a. idéia: indexar cada predicado por uma hash table, armazenar os valores V ou F na chave P caso tenhamos na base respectivamente P ou ¬P
  - Problema: como tratar sentenças complexas, variáveis?

Sistemas Baseados em Raciocínio Lógico

- 2a. Idéia: armazenar cada predicado com uma chave, contendo 4 componentes:
  - Lista de literais positivos
  - Lista de literais negativos
  - Lista de sentenças com predicado do lado esquerdo
  - Lista de sentenças com predicado
- Com isto, a implementação é eficiente para encadeamento progressivo ou regressivo
- Eventualmente, pode-se depois fazer uma nova representação de acordo com os valores das variáveis, montando uma árvore

Sistemas Baseados em Raciocínio Lógico

- Exemplo:

Brother(Richard, John)
Brother(Ted, Jack) ∧Brother(Jack, Bobbie)
¬Brother(Ann, Sam)
Brother(x,y) → Male(x)
Brother(x,y) ∧ Male(y) → Brother(y,x)
Male(Jack) ∧ Male(Ted) ∧... ∧ ¬ Male(Ann)

Sistemas Baseados em Raciocínio Lógico

Chave	Positiva	Negativa	Conclusão	Premissa
Brother	Brother(Richard,John) Brother(Ted, Jack) Brother(Jack, Bobbie)	¬Brother(Ann,Sam)	Brother(x,y) ∧ Male(y) → Brother(y,x)	Brother(x,y) ∧ Male(y) → Brother(y,x) Brother(x,y) → Male(y)
Male	Male(Jack) Male(ted) …	¬Male(Ann) …	Brother(x,y) → Male(y)	Brother(x,y) ∧ Male(y) → Brother(y,x)

Linguagens de Programação Lógica

- Prolog é o exemplo mais conhecido
- Kowalski: "Algorithm = Logic + Control"
- Programa = sequência de cláusulas de Horn
- Negação por falha : **`not P`** é considerado provado caso o programa falhe em provar **`P`**
- Predicados built-in para aritmética, entrada e saída (ex: `X is 3 + 4`)

Linguagens de Programação Lógica

- Exemplo:

∀x ∀l Member(x, [x|l])

∀x ∀y ∀x Member(x, l) → Member(x,[y|l])

- Em Prolog:

```
member(X, [X|L]).
member(X,[Y|L]) :- Member(X, L).
```

Linguagens de Programação Lógica

- As inferências de Prolog são realizadas com encadeamento regressivo, utilizando busca em profundidade
  - Inferência incompleta, cabe aos programadores se preocupar em não utilizar definições recursivas infinitas
- A ordem de busca é da esquerda para a direita para os conjuntos de uma premissa, e do início para o final para as cláusulas da BC
- A rotina de unificação não realiza a verificação de ocorrências internas (occur-check)
  - Inferência não correta, mas erros ocorrem muito raramente na prática

Provadores de Teoremas

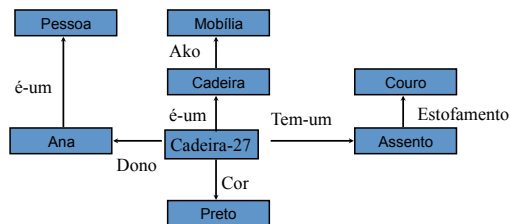
- Diferem das linguagens de programação lógica
  - Aceitam quaisquer sentenças de lógica de primeira ordem, e não apenas cláusulas de Horn
  - Não existe interferência entre lógica e controle. Por exemplo, B ∧ C → A, C ∧ B → A, C ∧ ¬A → ¬B dão o mesmo resultado
- Geralmente, dividem o conhecimento entre
  - *Cláusulas de suporte*: sempre utilizadas na resolução
  - *Axiomas*: conhecimento sobre o domínio
  - *Demoduladores*: simplificam expressões; por exemplo, se temos x+0=x, substituem as ocorrências de x+0 por x.

Sumário

1. Introdução
2. Sistemas de Produção
3. Sistemas Baseados em Lógica
4. Redes Semânticas
5. Frames
6. Sistemas de Manutenção da Verdade (TMS)
7. Conclusões

Redes Semânticas

- Histórico
  - Redes Semânticas foram propostas em 1913 por Selz como uma explicação a fenômenos psicológicos.
  - Em 1966, Quillian implementou essas redes e mostrou como o conhecimento semântico poderia ser representado como relacionamento entre dois objetos.
- Uma rede semântica é uma representação na qual
  - existem nós que representam entidades e links (predicados) que representam relacionamentos entre essas entidades;
  - cada link conecta um nó origem até um nó destino;
  - normalmente, os nós e links denotam entidades de domínio específico.

Exemplo: Rede Semântica

Redes Semânticas

- Forma mais flexível e intuitiva de representar conhecimento.
- Suportam herança de propriedades.
- Relações
  - **Ako (a-kind-of):** relações entre classes
  - **é-um (is-a):** relações entre classes e instâncias
    - uma entidade pertence a uma classe mais alta ou uma categoria de objetos.
  - **tem-um (has-a):** identifica características ou atributos das entidades
  - **parte-de (part-of)**: identifica características ou atributos das entidades
  - **variados:** identifica características gerais

Sistemas de Redes Semânticas

- Base de conhecimento
  - nós e links da rede.
- Máquina de inferência
  - busca e casamento de padrões
  - a busca se dá para frente e para trás através dos links.
- A busca pode ser usada de várias maneiras para se extrair informações
  - como uma ferramenta explicativa;
  - para explorar exaustivamente um tópico;
  - para encontrar o relacionamento entre dois objetos.

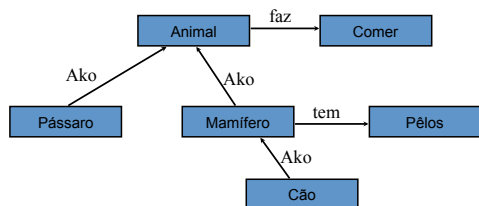
Exemplo: Busca em redes semânticas

Animal — faz → Comer

Pássaro — Ako → Animal

Mamífero — Ako → Animal

Mamífero — tem → Pêlos

Cão — Ako → Mamífero

Busca como Ferramenta Explicativa

- Para provar a declaração "Cães comem"
  - pode-se supor que cães comem, e usar busca sobre a rede para provar a hipótese.
- Buscando a partir do nó "Cão", temos:
  - "Cão **é-um** mamífero"
  - "Mamífero **é-um** animal"
  - "Animal **faz** comer"
  - Isto é uma prova para "Cães comem"

Explorar exaustivamente um tópico

- Para derivar todo o conhecimento sobre "cães", usa-se Busca em Largura a partir do nó "Cão"
  - "Cães são Mamíferos"
  - "Cães têm Pêlos"
  - "Cães são Animais"
  - "Cães Comem"

Relacionando tópicos

- Para verificar se "Cães" e "Pássaros" estão relacionados, pode-se executar, a partir de ambos os nós, uma Busca em Largura.
- A interseção entre os nós visitados nos dá uma pista sobre o relacionamento entre os nós iniciais.
- Isto é chamado *ativação distribuída* ou *interseção de busca*.

Vantagens

- Representação visual fácil de entender.
- Flexibilidade na manipulação de nós e links
  - adição, exclusão, modificação
- Economia
  - herança via relações "**é-um**" e "**ako**".
- Capta "senso-comum"
  - semelhante ao armazenamento de informações no cérebro.

Limitações

- Busca em redes semânticas grandes pode ser muito ineficiente.
- Não há homogeneidade na definição de nós e links.
- Hereditariedade pode causar dificuldades no tratamento de exceções.
- Pode haver conflito entre características herdadas.
- É difícil representar conhecimento procedimental
  - seqüenciamento e tempo não estão explícitos.
- Menos expressiva que a Lógica de Primeira Ordem
  - não há quantificadores.

Sumário

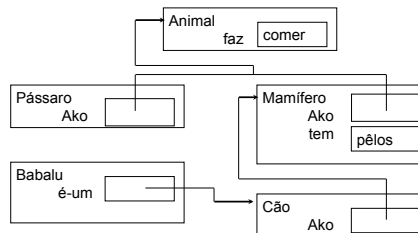
1. Introdução
2. Sistemas de Produção
3. Sistemas Baseados em Lógica
4. Redes Semânticas
5. Frames
6. Sistemas de Manutenção da Verdade (TMS)
7. Conclusões

Frames (quadros)

- Histórico
  - Artigos publicados por Minsky (1974), Winston (1975) Haugeland (1981), Brachman e Levesque (1985)
- Características
  - Um frame é identificado por um nome e descreve um objeto complexo através de um conjunto de atributos
  - Um **Sistema de Frames** é um conjunto de frames organizados hierarquicamente.
  - São uma evolução das Redes Semânticas:
    - nós são substituídos por frames
    - arcos são substituídos por atributos (slots)
    - **procedimentos podem ser anexados a um frame**

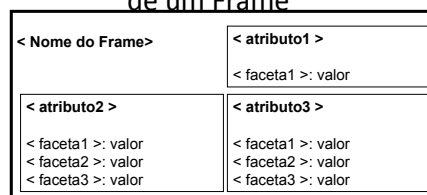
Frames: atributos (slots)

- Frames
  - Possuem pelo menos dois atributos:
    - *Nome*
    - Ako ou is-a
  - A fim de melhorar a estruturação (hierarquia), privilegiam dois tipos de relações:
    - ako: relação entre classe e sub-classe
    - is-a: relação entre classe e instância.
- Cada atributo
  - aponta para um outro frame ou para um tipo primitivo, ex. *string*;
  - consiste em um conjunto de facetas (atributos de atributos).

Exemplo: Classes e Instâncias

Facetas

- Descrevem conhecimento ou algum procedimento relativo ao atributo.
- Propriedades
  - **Valor**: especifica o único valor possível.
  - **Valor default:** especifica o valor assumido pelo atributo caso não haja nenhuma informação a esse respeito.
  - **Tipo**: indica o tipo de dado do valor.
  - **Domínio**: descreve os valores possíveis para o atributo.
  - **Procedimentos Daemons**
    - como os triggers nos bancos de dados

Uma Representação Abstrata de um Frame

< Nome do Frame>	**< atributo1 >** < faceta1 >: valor
< atributo2 > < faceta1 >: valor < faceta2 >: valor < faceta3 >: valor	**< atributo3 >** < faceta1 >: valor < faceta2 >: valor < faceta3 >: valor

- Os frames integram conhecimento **declarativo** sobre objetos e eventos e conhecimento **procedimental** sobre como recuperar informações ou calcular valores.

Procedimentos Daemons

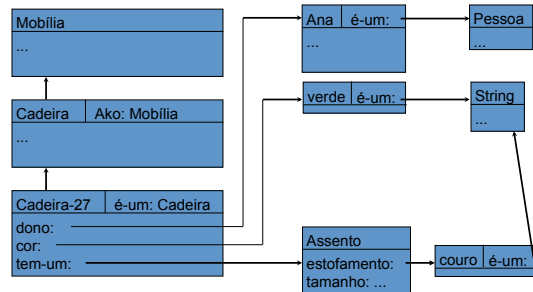
- Definição
  - São procedimentos *anexados* aos frames, disparados por consultas ou atualizações.
  - Podem inferir valores para atributos a partir de valores de outros atributos especificados anteriormente em qualquer frame do sistema.
- Procedimentos Daemons:
  - when-requested
    - quando o valor é pedido mas não existe ainda
  - when-read
    - quando valor é lido
  - when-written
    - quando valor é modificado

Exemplo: Procedimentos Daemons

Cômodo	Ako: Lugar-coberto		
Atributo	Default	Tipo	Se-necessário
Nº de paredes	4	número	
Formato	retangular	símbolo	
Altura	3	número	
Área		número	
Volume		número	Área * Altura

↑ Ako

Sala	Ako: Cômodo	
Atributo	Default	Tipo
Mobiliário	(sofá,mesa,cadeiras)	lista de símbolos
Finalidade	convivência	símbolo
Área	25	número

Exemplo de Sistema de Frames

Herança de Propriedades

- Três tipos de informações podem ser de herdadas
  - valor (= POO)
  - procedimento (= POO)
  - valor default
- Idéia: herdar das classes superiores
  - em caso de conflito, vale a informação mais específica
- Existem dois tipos de herança:
  - **Herança simples**
    - existe uma única super-classe para cada classe
  - **Herança múltipla**
    - uma classe pode ter mais de uma super-classe, podendo herdar propriedades ao longo de diversos caminhos diferentes (= o caos)

Sistemas Frames: Funções (historicamente)

- Reconhecer que uma dada situação pertence a uma certa categoria (matching)
  - ex. reconhecimento visual de uma sala de aula
- Interpretar a situação e/ou prever o que surgirá em termos da categoria reconhecida (matching)
  - ex. pessoa com revolver (revolver arma -> perigo)
- Capturar propriedades de senso comum sobre pessoas, eventos e ações
  - foi a primeira tentativa de estruturar conhecimento declarativo sem usar regras.
  - Deu origem ao que chamamos hoje de **Ontologias**!

Sumário

1. Introdução
2. Sistemas de Produção
3. Sistemas Baseados em Lógica
4. Redes Semânticas
5. Frames
6. Sistemas de Manutenção da Verdade (TMS)
7. Conclusões

Sistemas Baseados em Lógica Descritiva

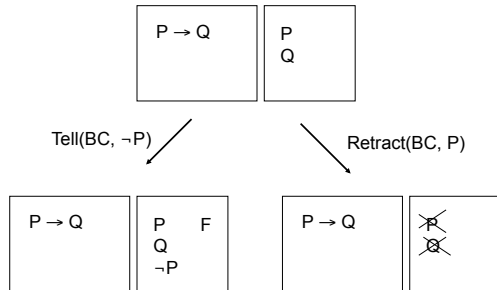
- Ao invés de falar sobre objetos, como em lógica de predicados, tais lógicas falam sobre categorias
- As principais metas da inferência são:
  - Subsumption: verificar se uma categoria é subconjunto de outra
  - Classificação: verificar se um objeto pertence a uma categoria
- Ex: Linguagem Classic

$\forall x$ Bachelor(x) $\leftrightarrow$ Unmarried(x) $\wedge$ Adult(x) $\wedge$ Male(x)

Bachelor = And(Unmarried, Adult, Male)

Sistemas de Manutenção da Verdade

- Existem ocasiões onde desejamos retirar fatos da BC:
  - O fato não é mais importante, e precisa-se de espaço em memória
  - O sistema está preocupado apenas com o estado corrente do mundo, e este muda
  - O sistema assumiu anteriormente que o fato era uma hipótese, e agora não deseja mais esta suposição
- Deseja-se que esta eliminação não traga inconsistências ao sistema
- Importante: Existe uma diferença entre realizar asa ações Tell(KB, $\neg$P) e Retract (KB,P)

Sistemas de Manutenção da Verdade

Sistemas de Manutenção da Verdade

- Este processo de de gerenciar quais termos e sentenças devem ser retirados da BC pelo fato de retirar um termo se chama manutenção da verdade (TMS)
- Servem para dar explicações das inferências
- JMTS: justification-based TMS
  - Cada sentença tem uma anotação associada
    Q: {P, P → Q}, S: {P, P ∨ R → S}, U: {R, P ∨ R→ U},
  - Servem para auxiliar na remoção de sentenças
    Se retiro P da BC, devo também retirar Q e S, mas não U
- ATMS: assumption-based TMS
  - Representam vários estados possíveis ao mesmo tempo
  - Q: {{P, P → Q}, {R, P ∨ R→ Q}}

Sumário

1. Introdução
2. Sistemas de Produção
3. Sistemas Baseados em Lógica
4. Redes Semânticas
5. Frames
6. Sistemas de Manutenção da Verdade (TMS)
7. Conclusões

Sistemas Baseados em Conhecimento

- Representação do Conhecimento (RC):
  - Linguagem: sintaxe, semântica, método de prova
  - Ontologia: sobre o que e como falar
  - Implementação: eficiência, representação, algoritmos
- Compromisso entre expressividade e eficiência
  - Completude x rapidez (ex: busca em profundiade em Prolog)
  - Corretude x simplificações (ex: occur-check em Prolog)
- Programação declarativa
  - O controle é praticamente built-in
  - Modular
  - Extensível
  - Comprensível

Referências Bibliográficas

- S. Russel and P. Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall, Upper Saddle River, USA. 2nd. Edition, 2003. Chapter 8 and 9.
- G. Bittencourt. Inteligência Artificial: Ferramentas e Teorias. Editora da UFSC, Florianópolis. 2a. Edição, 2001. Cap. 3.

Referências Bibliográficas

- Jackson, Peter. Introduction to Expert Systems. Second Edition. Addison-Wesley Publishing Company, 1990, p. 206-216
- Maida, Anthony S.. Encyclopedia of Artificial Intelligence. p. 493-507.
- Rich, Elaine; Knight, Kevin. Inteligência Artificial. Segunda Edição. Editora McGraw-Hill Ltda., 1993, p. 290-316
- Russel, Stuart; Norvig, Peter. Artificial Intelligence. A Modern Approach. Prentice-Hall Inc., 1995, p. 316-327
- Sowa, J.. Encyclopedia of Artificial Intelligence. p. 1011-1024.
- Winston, Patrick Henry. Artificial Intelligence. Third Edition. Addison-Wesley Publishing Company, 1992, p. 179-209